

日本国特許庁
JAPAN PATENT OFFICE

JPO
10/043564
01/11/02

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application:

2001年 1月11日

出願番号
Application Number:

特願2001-004189

出願人
Applicant(s):

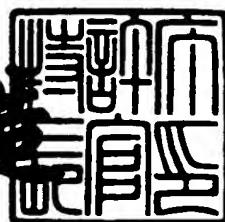
インターナショナル・ビジネス・マシーンズ・コーポレーション

CERTIFIED COPY OF
PRIORITY DOCUMENT

2001年 6月20日

特許庁長官
Commissioner,
Japan Patent Office

及川耕造



出証番号 出証特2001-3058440

【書類名】 特許願
【整理番号】 JP9000230
【提出日】 平成13年 1月11日
【あて先】 特許庁長官 殿
【国際特許分類】 G06F 17/30
【発明者】
【住所又は居所】 神奈川県大和市下鶴間1623番地14 日本アイ・ビ
ー・エム株式会社 東京基礎研究所内
【氏名】 渋谷 哲朗
【特許出願人】
【識別番号】 390009531
【氏名又は名称】 インターナショナル・ビジネス・マシーンズ・コーポレ
ーション
【代理人】
【識別番号】 100086243
【弁理士】
【氏名又は名称】 坂口 博
【代理人】
【識別番号】 100091568
【弁理士】
【氏名又は名称】 市位 嘉宏
【代理人】
【識別番号】 100106699
【弁理士】
【氏名又は名称】 渡部 弘道
【復代理人】
【識別番号】 100104880
【弁理士】
【氏名又は名称】 古部 次郎

【選任した復代理人】

【識別番号】 100100077

【弁理士】

【氏名又は名称】 大場 充

【手数料の表示】

【予納台帳番号】 081504

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9706050

【包括委任状番号】 9704733

【包括委任状番号】 0004480

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 パターン検索方法、パターン検索装置、コンピュータプログラム及び記憶媒体

【特許請求の範囲】

【請求項1】 検索対象である文字列中から所望のパターンを検索するパターン検索方法において、

前記パターンの最後の文字から前方へ1文字ずつ順に加えて得られる各中間パターンに関して、当該中間パターンの先頭の文字が前記文字列に対する接尾辞配列のどの範囲に存在するかを順次検索する範囲検索ステップと、

前記検索により前記パターン 자체に関して得られた前記接尾辞配列の前記範囲に含まれる各要素に対応する前記文字列の要素を特定し、当該文字列の各要素を先頭として前記パターンの要素数と同じ数の要素からなる部分文字列を抽出する文字列抽出ステップと

を含むことを特徴とするパターン検索方法。

【請求項2】 前記範囲検索ステップは、

前記文字列に対する接尾辞配列の各要素に関して、当該各要素に対応する前記文字列中の各文字の一つ前に位置する前置文字を特定するステップと、

前記接尾辞配列中の所定の要素以前の各要素における前記前置文字の中に含まれる、前記パターン中の所望の文字の個数を求めるステップと、

求められた前記文字の個数に基づいて、当該文字が前記接尾辞配列のどの位置に存在するかを検出するステップと

を含むことを特徴とする請求項1に記載のパターン検索方法。

【請求項3】 検索対象である配列中から所望のパターンを検索するパターン検索方法において、

前記パターンの最後の要素が前記配列中のどこに位置するかを検索するステップと、

前記パターンが複数の要素により構成されている場合に、当該パターン中の前記最後の要素に当該最後の要素の前に位置する要素を後ろから順に一つずつ加えて得られる各中間パターンに関して、当該中間パターンが前記配列中のどこに位

置するかを順次検索するステップと
を含むことを特徴とするパターン検索方法。

【請求項4】 検索対象である文字列中から所望のパターンを検索するパターン検索方法において、

前記パターンの最初の文字から後方へ1文字ずつ順に加えて得られる各中間パターンについて、当該中間パターンの最後の文字が前記文字列に対する接頭辞配列のどの範囲に存在するかを順次検索する範囲検索ステップと、

前記検索により前記パターン自身について得られた前記接頭辞配列の前記範囲に含まれる各要素に対応する前記文字列の要素を特定し、当該文字列の各要素を最後尾として前記パターンの要素数と同じ数の要素からなる部分文字列を抽出する文字列抽出ステップと

を含むことを特徴とするパターン検索方法。

【請求項5】 検索対象である遺伝子配列中から所望のパターンを検索するパターン検索方法において、

前記パターンの最後の要素から前方へ一つずつ順に加えて得られる各中間パターンについて、当該中間パターンの先頭の要素が前記遺伝子配列に対する接尾辞配列のどの範囲に存在するかを順次検索する範囲検索ステップと、

前記検索により前記パターン自身について得られた前記接尾辞配列の前記範囲に含まれる各要素に対応する前記遺伝子配列の要素を特定し、当該遺伝子配列の各要素を先頭として前記パターンの要素数と同じ数の要素からなる部分配列を抽出する配列抽出ステップと

を含むことを特徴とするパターン検索方法。

【請求項6】 検索対象である文字列中から所望のパターンを検索するパターン検索装置において、

前記文字列の接尾辞配列に基づいて前記パターンを検索するためのデータ構造を構築する前処理部と、

前記前処理部により構築されたデータ構造を用いて所望の前記パターンを検索する検索部とを備え、

前記前処理部は、

前記接尾辞配列の各要素に関して、当該各要素に対応する前記文字列中の各文字の一つ前に位置する前置文字を特定し、

前記接尾辞配列中の所定の要素以前の各要素における前記前置文字の、前記文字列を構成する文字の種類ごとの個数を求めるこにより前記データ構造を構築することを特徴とするパターン検索装置。

【請求項7】 前記前処理部は、前記前置文字の個数を、前記接尾辞配列における要素の位置と、前記文字列を構成する文字の種類とに対応付けて格納したテーブルを持つことを特徴とする請求項6に記載のパターン検索装置。

【請求項8】 前記前処理部は、前記接尾辞配列の所定数個おきの要素の位置に関して生成された前記テーブルを持つことを特徴とする請求項7に記載のパターン検索装置。

【請求項9】 前記前処理部は、前記テーブルにおいて情報が管理される前記接尾辞配列の所定の位置に基づいて、当該位置の間における前記接尾辞配列の要素に対する前記前置文字に関する情報を格納した他のテーブルをさらに持つことを特徴とする請求項8に記載のパターン検索装置。

【請求項10】 前記検索部は、

前記接尾辞配列中の所定の要素以前の各要素における前記前置文字の個数に基づいて、前記パターンの最後の文字から前方へ1文字ずつ順に加えて得られる各中間パターンに関して、当該中間パターンの先頭の文字が前記文字列に対する接尾辞配列のどの範囲に存在するかを順次検索し、

前記検索により前記パターン自体に関して得られた前記接尾辞配列の前記範囲に含まれる各要素に対応する前記文字列の要素を特定し、当該文字列の各要素を先頭として前記パターンの要素数と同じ数の要素からなる部分文字列を抽出することを特徴とする請求項6に記載のパターン検索装置。

【請求項11】 コンピュータに、検索対象である配列中から所望のパターンを検索する処理を実行させるコンピュータプログラムにおいて、

前記パターンの最後の要素から前方へ一つずつ順に加えて得られる各中間パターンに関して、当該中間パターンの先頭の要素が前記配列に対する接尾辞配列のどの範囲に存在するかを順次検索する処理と、

前記検索により前記パターン自体に関して得られた前記接尾辞配列の前記範囲に含まれる各要素に対応する前記配列の要素を特定し、当該配列の各要素を先頭として前記パターンの要素数と同じ数の要素からなる部分配列を抽出する処理とを前記コンピュータに実行させることを特徴とするコンピュータプログラム。

【請求項12】 コンピュータに、検索対象である配列中から所望のパターンを検索する処理を実行させるコンピュータプログラムにおいて、

前記検索対象である配列に対する接尾辞配列の各要素に関して、当該各要素に対応する前記配列中の各要素の一つ前に位置する前置要素を特定する処理と、

前記接尾辞配列中の所定の要素以前の各要素における前記前置要素の、前記配列を構成する要素の種類ごとの個数を求める処理と、

前記接尾辞配列中の所定の要素以前の各要素における前記前置要素の個数に基づいて、前記パターンの最後の要素から前方へ一つずつ順に加えて得られる各中間パターンに関して、当該中間パターンの先頭の要素が前記配列に対する接尾辞配列のどの範囲に存在するかを順次検索する処理と、

前記検索により前記パターン自体に関して得られた前記接尾辞配列の前記範囲に含まれる各要素に対応する前記配列の要素を特定し、当該配列の各要素を先頭として前記パターンの要素数と同じ数の要素からなる部分配列を抽出する処理とを前記コンピュータに実行させることを特徴とするコンピュータプログラム。

【請求項13】 前記前置要素の個数を、前記接尾辞配列における要素の位置と、前記配列を構成する要素の種類とに対応付けて格納したテーブルを生成し、記憶装置に保持する処理を前記コンピュータにさらに実行させることを特徴とする請求項12に記載のコンピュータプログラム。

【請求項14】 コンピュータに検索対象である配列中から所望のパターンを検索する処理を実行させるプログラムを当該コンピュータの入力手段が読み取可能な記憶した記憶媒体において、

前記プログラムは、前記パターンを検索する処理として、

前記パターンの最後の要素から前方へ一つずつ順に加えて得られる各中間パターンに関して、当該中間パターンの先頭の要素が前記配列に対する接尾辞配列のどの範囲に存在するかを順次検索する処理と

前記検索により前記パターン自体に関して得られた前記接尾辞配列の前記範囲に含まれる各要素に対応する前記配列の要素を特定し、当該配列の各要素を先頭として前記パターンの要素数と同じ数の要素からなる部分配列を抽出する処理とを前記コンピュータに実行させることを特徴とする記憶媒体。

【請求項15】 コンピュータに検索対象である配列中から所望のパターンを検索する実行させるプログラムを当該コンピュータの入力手段が読み取可能に記憶した記憶媒体において、

前記プログラムは、前記パターンを検索する処理として、

前記接尾辞配列の各要素に関して、当該各要素に対応する前記配列中の各要素の一つ前に位置する前置要素を特定する処理と、

前記接尾辞配列中の所定の要素以前の各要素における前記前置要素の、前記配列を構成する要素の種類ごとの個数を求める処理と、

前記接尾辞配列中の所定の要素以前の各要素における前記前置要素の個数に基づいて、前記パターンの最後の要素から前方へ一つずつ順に加えて得られる各中間パターンに関して、当該中間パターンの先頭の要素が前記配列に対する接尾辞配列のどの範囲に存在するかを順次検索する処理と、

前記検索により前記パターン自体に関して得られた前記接尾辞配列の前記範囲に含まれる各要素に対応する前記配列の要素を特定し、当該配列の各要素を先頭として前記パターンの要素数と同じ数の要素からなる部分配列を抽出する処理とを前記コンピュータに実行させることを特徴とする記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、文字列などの配列中に存在する頻出部分配列や、二つ以上の配列に共通な部分配列を検索するためのデータ構造及びこのデータ構造を用いたパターン検索方法に関する。

【0002】

【従来の技術】

文字列中に存在する頻出部分文字列や、二つ以上の文字列に共通な部分文字列

などを高速で検索するのに有効なデータ構造として、接尾辞木 (Suffix tree) が知られている。接尾辞木は、処理対象の文字列中に存在しない文字 \$ を処理対象の文字列の最後に加えた文字列における全ての接尾辞を表す木である。接尾辞木の葉ノード（各枝において他の枝が接続されていない先端のノード）は、それぞれの接尾辞に対応する。

ここで、接尾辞とは、所定の文字列において、所定の文字を特定した場合の当該文字以降の文字列である。

図6は、接尾辞木の例を示す図である。図6には、処理対象の文字列として「mississippi」の最後に文字 \$ を加えた文字列「mississippi \$」の接尾辞木を示す。

【0003】

接尾辞木において、各枝は、部分文字列に相当するラベルを持つ。そして、ルートノードから葉ノードまでの各枝が持つラベルを並べたものが、当該葉ノードに対応する接尾辞となる。図6に示す例では、例えば、ルートノードから「i」 「ss」 「pp」 のラベルを持つ各枝を経て到達する葉ノードに対応する接尾辞は「issippi」であり、同様に「s」 「si」 「ssippi」 のラベルを持つ各枝を経て到達する葉ノードに対応する接尾辞は「ssiissippi」である。

【0004】

また、接尾辞木における单一のノード（ルートノードを含む）から出てゆく各枝に付されているラベルの最初の文字は全て異なり、これらはラベルの最初の文字でソートされている。図6に示す例では、図の左側から右側へ向けて英語のアルファベット順（i, m, p, s の順）に枝が並んでいる。

【0005】

接尾辞木を生成するアルゴリズムとしては、処理対象である文字列の長さを n 、文字列を構成するアルファベットのサイズ（文字の種類の数）を s とした場合、 $O(n \log s)$ のアルゴリズムが知られている。特にアルファベットが整数アルファベット（1から n までの数字）である場合は、 $O(n)$ のアルゴリズムが知られている。ここで、 $O(func(n))$ は、実際の計算時間が t である場合に

、 $n \geq k$ であるような n に対して、

$$0 \leq t \leq c \times \text{func}(n)$$

が成り立つような何らかの定数 c と k の組が必ず存在することを意味する。したがって、 $O(n \log s)$ は $n \log s$ の定数倍以内の時間で計算が可能であることを意味し、 $O(n)$ は n の定数倍以内の時間（この場合、 n も定数なので、定数時間内）で計算できることを意味する。

【0006】

これを用いれば、長さ m の部分文字列の検索は、 $O(m \log s)$ に相当する時間で行うことができる。通常、アルファベットのサイズは定数サイズなので、この時間は線形時間といって良い。

英文字テキスト（ n 文字）に対するこの接尾辞木を扱うために必要とする記憶装置の記憶容量は、 $20n$ バイト～ $40n$ バイトである。

【0007】

この接尾辞木のデータサイズは大きいため、このデータサイズを抑制する類似のパターン検索用のデータ構造として、接尾辞配列（suffix array）が知られている。

上述したように、接尾辞木の葉ノードは、それぞれが文字列の接尾辞に対応している。この接尾辞を、接尾辞木の一端側（図6の例では左端側）の葉ノードに対応する接尾辞から順に並べると、処理対象の文字列における全ての接尾辞を辞書的順序で並べた配列が得られる。ただし、各接尾辞は、最後に終了判定文字 \$ を付加されているものとする。

【0008】

この配列の構成要素である各接尾辞を、処理対象の文字列における当該接尾辞の最初の文字の位置を表す情報で置き換える（例えば、「ippi\$」を「8」に、「issippi\$」を「5」にというように置き換える）。これにより、処理対象の文字列と同じ長さの配列（接尾辞配列）が得られる。例えば、図6における「mississippi\$」の接尾辞配列は、「8 5 2 11 1

10 9 7 4 6 3 12」となる。なお、文字\$は他の全ての文字よりも辞書的順序が後であるとしている。

【0009】

この接尾辞配列を用いると、接尾辞木を用いる場合と比較して、文字列検索を行うために必要なメモリ容量を削減することができる。また、文字列の検索に要する時間は、2分探索を行うことから、 $O(p \log q)$ となる。ただし、qはデータベースの大きさ、pは検索しようとする文字列の長さである。

通常、必要な記憶容量は一つの文字に対し4バイトであるから、テキストが英文字(1バイト)の場合、n文字のテキストに対するこのデータベースのデータサイズは5nバイトである。

【0010】

また、データベースに、さらに隣接する接尾辞の共通接頭辞長のテーブルを持たせることもできる。このテーブルを用いると、接尾辞木配列のみを用いる場合に対して、検索時間を $O(p + \log q)$ と短縮することができる。なお、この場合におけるデータベースのデータサイズは9nバイトとなる。

【0011】

【発明が解決しようとする課題】

大規模なテキストデータベースを検索するために、上述した接尾辞木や接尾辞配列をデータ構造として用いる場合、次のような問題がある。

まず、接尾辞木をデータ構造として用いる場合、必要とされるデータベースの大きさが大きいという問題がある。

上述したように、処理対象である文字列の長さがnである場合、このテキストに対する接尾辞木を扱うために必要な記憶装置の記憶容量、すなわちデータベースのデータサイズは、20nバイト～40nバイトである。一般に、データ構造として接尾辞木を用いる場合、記憶装置に対して、接尾辞配列を用いる場合の4～6倍の記憶領域(接尾辞配列では、1バイト文字で文字数nのテキストの場合、5nバイト)を要する。

このため、大規模なテキストデータベースに対して接尾辞木を使用することは困難である。

【0012】

一方、接尾辞配列をデータ構造として用いる場合、検索に長時間を要するという問題がある。

接尾辞配列に対して検索を行う場合、2分探索を行うため、データベースの大きさを q 、検索しようとする文字列の長さを p として、 $O(p \log q)$ だけの時間をする。したがって、アルファベットのサイズが定数サイズである場合にほぼ線形時間で検索を行うことができる接尾辞木に比べて、多大な計算時間を要する。

また上述したように、データサイズが多少大きくなることを許し、データベースに、接尾辞配列中で隣接する接尾辞の共通接頭辞長のテーブルを持たせることによって、計算時間を $O(p + \log q)$ に短縮することができる。しかしこの場合であっても、依然として $\log q$ の項が残っているため、接尾辞木の場合と比べると、多大な計算時間を要する。

【0013】

そこで本発明は、大規模テキストデータベースの検索において、処理を行うためのデータ構造におけるデータサイズの増大を抑えながら、高速な検索を実現することを目的とする。

【0014】

【課題を解決するための手段】

かかる目的のもと、本発明は、検索対象である文字列中から所望のパターンを検索するパターン検索方法において、次の範囲検索ステップと、文字列抽出ステップとを含むことを特徴とする。すなわち、範囲検索ステップにおいて、このパターンの最後の文字から前方へ1文字ずつ順に加えて得られる各中間パターンに関して、この中間パターンの先頭の文字が検索対象の文字列に対する接尾辞配列のどの範囲に存在するかを順次検索する。この検索をパターンの最後の文字から順に実行することによって、最終的に、このパターン自身を含む接尾辞配列の範囲が求められる。次に、文字列抽出ステップにおいて、当該接尾辞配列の範囲に含まれる各要素に対応する文字列の要素を特定し、この文字列の各要素を先頭としてこのパターンの要素数と同じ数の要素からなる部分文字列を抽出する文字列

抽出ステップとを含むことを特徴とする。

上記のように構成されたパターン検索は、アルファベットや日本語のテキストなど種々の文字による文字列における検索に用いることができるが、バイナリデータや遺伝子配列のような使用される文字の種類が少ない文字列から所望のパターンを検索する場合には、検索に用いるデータ構造のデータサイズを特に小さくすることができる。

【0015】

ここで、この範囲検索ステップは、検索対象の文字列に対する接尾辞配列の各要素について、この各要素に対応する文字列中の各文字の一つ前に位置する前置文字を特定するステップと、接尾辞配列中の所定の要素以前の各要素における前置文字の中に含まれる、このパターン中の所望の文字の個数を求めるステップと、求められた文字の個数に基づいて、この文字が接尾辞配列のどの位置に存在するかを検出するステップとを含むことを特徴とする。

【0016】

また、本発明は、検索対象である配列中から所望のパターンを検索するパターン検索方法において、このパターンの最後の要素が前記配列中のどこに位置するかを検索するステップと、このパターンが複数の要素により構成されている場合に、このパターン中の最後の要素に、この最後の要素の前に位置する要素を後ろから順に一つずつ加えて各中間パターンを得、この中間パターンが配列中のどこに位置するかを順次検索するステップとを含むことを特徴とする。

【0017】

また、本発明は、上記のように接尾辞配列を用いるパターン検索だけではなく、接頭辞配列を用いるパターン検索にも適用することができる。すなわち、上述した範囲検索ステップにおいて、このパターンの最初の文字から後方へ1文字ずつ順に加えて得られる各中間パターンについて、この中間パターンの最後の文字が検索対象の文字列に対する接頭辞配列のどの範囲に存在するかを順次検索する。この検索をパターンの最初の文字から順に実行することによって、最終的に、このパターン自体を含む接頭辞配列の範囲が求められる。次に、文字列抽出ステップにおいて、当該接頭辞配列の範囲に含まれる各要素に対応する文字列の要素

を特定し、この文字列の各要素を最後尾としてこのパターンの要素数と同じ数の要素からなる部分文字列を抽出する文字列抽出ステップとを含む構成とすることができる。

【0018】

また、本発明は、検索対象である文字列中から所望のパターンを検索するパターン検索装置において、この文字列の接尾辞配列に基づいてパターンを検索するためのデータ構造を構築する前処理部と、この前処理部により構築されたデータ構造を用いて所望のパターンを検索する検索部とを備え、この前処理部は、接尾辞配列の各要素に関して、この各要素に対応する文字列中の各文字の一つ前に位置する前置文字を特定し、接尾辞配列中の所定の要素以前の各要素における前置文字の、検索対象の文字列を構成する文字の種類ごとの個数を求ることによりデータ構造を構築することを特徴とする。

【0019】

ここで、前処理部は、前置文字の個数を、接尾辞配列における要素の位置と、検索対象の文字列を構成する文字の種類とに対応付けて格納したテーブルを持つことができる。

ここでさらに、このテーブルを、接尾辞配列の所定数個おきの要素の位置に関するテーブルとすることができます。すなわち、テーブルに格納するデータを間引くことにより、当該データ構造のデータサイズを縮小することができる。

さらにこの場合、前処理部は、間引いた範囲の前置文字の個数を算出する際に使用するため、このテーブルにおいて情報が管理される接尾辞配列の所定の位置に基づいて、この位置の間における接尾辞配列の要素に対する前置文字に関する情報を格納した他のテーブルをさらに持つことができる。

【0020】

また、このパターン検索装置において、検索部は、この接尾辞配列中の所定の要素以前の各要素における前置文字の個数に基づいて、パターンの最後の文字から前方へ1文字ずつ順に加えて得られる各中間パターンに関して、この中間パターンの先頭の文字が文字列に対する接尾辞配列のどの範囲に存在するかを順次検索する。この検索により、このパターン自体を含む接尾辞配列の範囲が得られる

。そして、この範囲に含まれる各要素に対応する文字列の要素を特定し、この文字列の各要素を先頭としてパターンの要素数と同じ数の要素からなる部分文字列を抽出する。

【0021】

また、本発明は、コンピュータに、検索対象である配列中から所望のパターンを検索する処理を実行させるコンピュータプログラムにおいて、このパターンの最後の要素から前方へ一つずつ順に加えて得られる各中間パターンに関して、この中間パターンの先頭の要素が検索対象である配列に対する接尾辞配列のどの範囲に存在するかを順次検索する処理と、この検索によりこのパターン自体に関して得られた接尾辞配列の範囲に含まれる各要素に対応する配列の要素を特定し、この配列の各要素を先頭としてこのパターンの要素数と同じ数の要素からなる部分配列を抽出する処理とをコンピュータに実行させることを特徴とする。

【0022】

さらにまた、本発明は、コンピュータに、検索対象である配列中から所望のパターンを検索する処理を実行させるコンピュータプログラムにおいて、検索対象である配列に対する接尾辞配列の各要素に関して、この各要素に対応する配列中の各文字の一つ前に位置する要素を特定する処理と、この接尾辞配列中の所定の要素以前の各要素における前置要素の、配列を構成する要素の種類ごとの個数を求める処理と、この接尾辞配列中の所定の要素以前の各要素における前置要素の個数に基づいて、このパターンの最後の要素から前方へ1文字ずつ順に加えて得られる各中間パターンに関して、この中間パターンの先頭の要素がこの配列に対する接尾辞配列のどの範囲に存在するかを順次検索する処理と、この検索によりこのパターン自体に関して得られた接尾辞配列の範囲に含まれる各要素に対応する配列の要素を特定し、この配列の各要素を先頭としてこのパターンの要素数と同じ数の要素からなる部分配列を抽出する処理とをコンピュータに実行させることを特徴とする。

これらのコンピュータプログラムは、例えば磁気ディスクその他の記憶媒体に格納して提供することができる。また、インターネットなどのネットワークを介して伝送させることにより提供することもできる。

【0023】

【発明の実施の形態】

以下、添付図面に示す実施の形態に基づいてこの発明を詳細に説明する。

図1は、本実施の形態を実現するのに好適なコンピュータ装置のハードウェア構成の例を模式的に示した図である。

図1に示すコンピュータ装置は、CPU（中央処理装置）101と、システムバスを介してCPU101に接続されたM/B（マザーボード）チップセット102及びメインメモリ103と、PCIバスなどの高速なバスを介してM/Bチップセット102に接続されたビデオカード104、ハードディスク105及びネットワークインターフェース106と、さらにブリッジ回路110及びISAバスなどの低速なバスを介してM/Bチップセット102に接続されたフロッピーディスクドライブ107、キーボード108及びシリアルI/Oポート109とを備える。

なお、図1は本実施の形態による検索方法を実現するコンピュータ装置の構成を例示するに過ぎず、本実施の形態を適用可能であれば、他の種々のシステム構成を取ることが可能である。

【0024】

本実施の形態は、図1に示したメインメモリ103に展開されたプログラムにてCPU101を制御することにより、所定の文字列（文字を要素とする配列）中から所望の部分文字列を検索する（以下、検索対象の文字列をテキスト、検索する部分文字列をパターンと称す）。

図2は、プログラム制御されたCPU101において、本実施の形態におけるデータ構造の構築及び検索を行うための機能ブロックを示す図である。

図2を参照すると、本実施の形態は、テキストの接尾辞配列を生成する接尾辞配列生成部10と、接尾辞配列生成部10にて生成された接尾辞配列を変換して所望のデータ構造を構築する前処理部20と、前処理部20にて構築されたデータ構造を用いてパターンの検索を行う検索部30とを備える。

【0025】

上述したように、これらの構成要素は、プログラム制御されたCPU101に

より実現される仮想的なソフトウェアブロックである。当該プログラムは、磁気ディスクや光ディスク、半導体メモリ、その他の記憶媒体に格納して提供したり、ネットワークを介して伝送したりすることができる。本実施の形態は、図1に示したネットワークインターフェース106やフロッピーディスクドライブ107、図示しないCD-ROMドライブなどを介して当該プログラムを入力し、ハードディスク105に格納する。そして、ハードディスク105に格納されたプログラムをメインメモリ103に読み込んで展開し、CPU101にて実行する。

【0026】

図2において、接尾辞配列生成部10は、図示しないデータベースから検索対象であるテキストを取得し、接尾辞配列を生成する。接尾辞配列の生成方法としては、公知の任意のアルゴリズムを用いることができる。検索対象であるテキストや生成された接尾辞配列は、メインメモリ103に格納される。なお、接尾辞配列は、公知の種々の方法により生成することが可能であり、外部装置において生成された接尾辞配列を本実施の形態において使用することもできる。したがって、接尾辞配列生成部10は必須の構成要素ではない。接尾辞配列生成部10を構成要素として設けない場合は、検索対象となるテキストと当該テキストの接尾辞配列とがメインメモリ103に直接格納されることとなる。

【0027】

以下の説明において、検索対象となるテキストをT[1・・・n]、検索するパターンをP[1・・・m]とする。また、テキストTに対する接尾辞配列をSA[1・・・n]とする。なお、以下の説明では、テキストTやパターンPの文字列はダブルクオーテーションマーク（“ ”）で囲み、その中の文字はクオーテーションマーク（‘ ’）で囲んで示すこととする。

例えば、「mississippi」の最後に文字\$を加えたテキストTは、

T[1・・・12] = “mississippi\$”

となる。ここで、‘\$’は、終了判定文字であり、辞書的順序が他の全ての文字よりも大きい（すなわち後に位置する）ものとする。また、テキストT[1・

・ 12] から3文字のパターン「s s i」を検索する場合、検索パターンPは、

$$P[1 \dots 3] = "s s i"$$

となる。さらにここで、テキストTに対し、 $T[5] = 'i'$ （5番目の文字）のように表すと、テキストTに対する接尾辞配列SAは、

$$\begin{aligned} SA[1 \dots 12] \\ = \{8 \ 5 \ 2 \ 11 \ 1 \ 10 \ 9 \ 7 \ 4 \ 6 \ 3 \ 12\} \end{aligned}$$

となる。

なお、終了判定文字「\$」は、概念上のものとして扱い、実際の処理においてはメインメモリ103に格納しなくても良い。この場合、 $T[i]$ にアクセスする際、 $i = 12$ ならば「\$」である、という分岐条件を入れることとなる。例えば、テキストTの文字数が256個あり、「\$」も1文字として扱うと1バイトに収まらなくなってしまうような場合は、メインメモリ103に格納しない方が望ましい。

【0028】

前処理部20は、接尾辞配列生成部10により生成された接尾辞配列SAを読み、これに基づいて、検索対象である文字列から $f(i, c)$ （ i はn以下の正の整数、 c は文字）で定義されるデータを検出するためのデータ構造を構築する。

ここで、 $f(i, c)$ は、 $T[SA[j] - 1] = c$ ($j \leq i$) であるような j の数である。配列Bを考え、 $B[i] = T[SA[i] - 1]$ とする。すなわち、配列Bは、接尾辞配列SAの各要素に対応するテキストTの各文字の一つ前に位置する文字（前置文字）の配列である。例えば、 $B[4] = T[SA[4] - 1] = T[10] = 'i'$ となる。同様にして配列Bの全ての文字を書き出すと次のようになる。

$B[1 \dots 12] = "ssmp\$pissi"$

したがって、上記の $f(i, c)$ の値は、配列 B において、インデックスが i 以下の文字 c の個数で表現することができる。例えば、 $f(6, 's') = 2$ であり、 $f(6, 'p') = 2$ であり、 $f(6, 'm') = 1$ である。なお、 $i > n$ であるような i に対しては、 $f(i, c) = f(n, c)$ と定義する。また、 $i \leq 0$ であるような i に対しては、 $f(i, c) = 0$ と定義する。

【0029】

$f(i, c)$ のデータ全体をテーブルとして保持すれば、パラメータである i 、 c を与えれば直ちに対応する $f(i, c)$ を求めることができる。しかし、テキスト T を構成する文字の種類 (s) が極めて少ない場合、例えばバイナリデータの文字列 (2種類: 0, 1) やDNA配列 (4種類: アデニン (A)、チミン (T)、グアニン (G)、シトシン (C)) では可能であるが、文字の種類 (s) が多い場合は、当該テーブルは極めて大きな配列となるため、現実的ではない。

そこで、前処理部20は、以下のようにして $f(i, c)$ を算出するためのデータ構造を構築する。

【0030】

(1) テーブル F の作成

k を適当な大きさの n 以下の正の整数であるとする。まず、すべての正の整数 i ($k * i < n + k$) に対して、 $f(k * i, c)$ のテーブルを作成する。これは、テキスト T を k 個の文字ごとに区切り、 k 番目の文字ごとに $f(i, c)$ を求めてテーブルを作成することに相当する。 $f()$ の大きさは n 以下であるから、このサイズは $(n * \log n) / k$ ビットである。これは、 n が 1 ワードに入る通常のケースでは $O(n * s / k)$ ワード (すなわち、 $n * s / k$ の定数倍以内) のことである。

このテーブルを F とし、

$F[i][c] = f(k * i, c)$

とする。なお、このテーブルFは、テーブルの大きさとテキストの大きさのうち大きい方に比例した時間で構築することができる。

【0031】

このテーブルFを持つことにより、 k の倍数のインデックスに関しては、 f を $O(1)$ の時間で求めることができる。そこで次に、 k の倍数以外のインデックスに関して f の値を求めるためのデータ構造を考える。

そのため、 $g(i, c, j)$ を、 $T[SA[p] - 1] = c$ を満たす p （ただし、 $k * (i - 1) < p \leq k * i$ ）のうち、 j 番目のものとし、まず、この $g(i, c, j)$ を求めるためのデータ構造について述べる。

【0032】

(2) テーブルLの作成

$h(i, c)$ を、 $f(k * i, c) - f(k * (i - 1), c)$ とする。これはテーブルFから直ちに計算可能である。

$l(i, c)$ を、 $h(i, d)$ ($d < c$, 辞書順) の総和として、これをテーブルとして持つ。このテーブルをLとし、

$$L[i][c] = l(i, c)$$

とする。このテーブルLのサイズは $(n * s \log k) / k$ ビットである。

【0033】

(3) テーブルGの作成

次に、全ての r ($0 < k * r < n + k$) に対して、 $0 < q \leq k$ を満たす整数 q を、 $T[SA[q + k * (r - 1)] - 1]$ の値が同じ物ごとに辞書的順序にしたがって並べ替えたものをテーブル $G[r][1 \dots k]$ とする。このとき、 $T[SA[q + k * (r - 1)] - 1]$ の値における辞書的順序が同じものに関しては q の値が小さいものが先になるように並べる。ただし、

【数1】

$$r = \lceil n/k \rceil$$

の場合、 $0 < q \leq n - (r - 1)k$ のような q だけを並べる。これは、数1を満足する r の範囲に含まれる文字の数が k 個に満たない場合があるためである。すなわち、上述したようにテーブルFの作成において、テキストTを k 個の文字ごとに区切ったが、テキストTの文字数 n が k で割り切れない場合は、最後尾の区分における文字数は k 個に満たない。したがって、 $0 < q \leq n - (r - 1)k$ のような q を並べることとする。

テーブルGの配列のサイズは全体で n であり、ビットで表すと $n \log k$ ビットということになる。これは、例えば次のようにして求めることができる。

```

for allc (すべての文字) w(c)=1;
for (q= 1; q<=k; q++) {
  c = T[SA[q+k*(r-1)]-1];
  G[r][L[r,c]+w(c)] = q;
  w(c) = w(c)+1;
}

```

ただし、 r の値が上記数1を満足する場合、forループは $(q = 1 ; q \leq n - (r - 1)k ; q++)$ となる。

【0034】

(4) $f(i, c)$ の計算

上記のようにして作成されたテーブルG及びテーブルLを用いて、 $g(i, c, j)$ を示すと、

$$g(i, c, j) = G[i] [L[i][c] + j] + k * (i - 1)$$

である。したがって、 $g(i, c, j)$ は G 、 L の二つテーブルから $O(1)$ 時間で得ることができる。

次に、 $k * (j - 1) < j \leq k * i$ であるような j に対し、 $T[SA[p] - 1] = c$ (ただし $k * (i - 1) < p \leq j$) となるような p の数を $f'(j, c)$ とする。そして、 $x (0 < x \leq h(i, c))$ の区間で、 $g(i, c, x)$ の値が j 以下となるような最大の x を見つけ出すと、

$$f'(j, c) = x$$

となる。この x の値は、 $g(i, c, x)$ の値が昇順になっているため、2分探索により $O(\log h(i, c))$ で計算可能である。 $h(i, c) < k$ であるから、これは、 $O(\log k)$ ということである ($h(i, c)$ の平均値は k/s であるため、実際にはより短い時間で計算できる)。ただし、このような x が存在しない場合は、 $f'(j, c) = 0$ とする。

以上の前提で、 $f(j, c)$ は、

$$f(j, c) = F[i - 1][c] + f'(j, c)$$

と計算できる。したがって、 $f(j, c)$ は、以上のデータ構造を用いることにより、 $O(\log k)$ で計算することができる。

【0035】

上述したテーブル F 、 L 、 G を表すのに必要なビット数は、テーブル F が $(n * s \log n) / k$ ビット、テーブル L が $(n * s \log k) / k$ ビット、テーブル G が $n \log k$ ビットであるから、全体で

$$(n * s / k) * (\log n + \log k) + n \log k \text{ ビット}$$

である。これらのテーブルF、L、Gは、メインメモリ103に格納される。

実際の運用においては、メインメモリ103の記憶容量として、これに加えて接尾辞配列SAのための $n \log n$ ビット及びテキストT自身のための $n \log s$ ビットが必要になる。

【0036】

また、前処理部20は、テキストTに関して、これらのデータ構造に加えて次に示すテーブルCも持つこととする。このテーブルの要素C[c]は、テキストTに含まれるc以下の文字の総数を表す。ただし、c以下の文字とは、cあるいはcより辞書的順序で早い文字を意味する。

テーブルCも他のデータ構造と同様に、メインメモリ103に格納される。このテーブルCのサイズは $s \log n$ ビットである。また、テーブルCはテキストTに対し、線形時間で計算可能である。

なお、kの値を小さく設定した場合には、 $j = i * k + d$ ($d < k$) に対して、 $f(i * k, c)$ を求める際、テーブルL、Gは持たずに、テーブルFから求められる $f(i * k, c)$ の値と、 $i * k + 1$ に基づいて、 $T[SA[j] - 1]$ の値がcであるものの個数を数えるという方法も考えられる。この場合の計算時間は $O(k)$ であるので、kと $\log k$ の値が近いような小さなkに対しては有効である。この方法を用いる場合は、テーブルL、Gを持たない分、必要なメモリの記憶容量は減少する。

【0037】

検索部30は、前処理部20にて作成された上記のデータ構造を用いて、テキストTから所望のパターンPを検索する。

検索は、 $f(i, c)$ を用い、次のように行う。

```

start = C[P[m]-1]+1 ;
end = C[P[m]] ;
for each i (m-1>=i>=1, 降順) {
    c = P[i] ;
    start = C[c-1] + f(start, c) ;

```

```

end = C[c-1] + f(end, c);
if (end < start) {
    パターンは存在しないので終了。
}
}

```

ただし、文字 c に対し、 $c + 1$ とは、辞書的順序で文字 c の次に来る文字を表し、 $c - 1$ は辞書的順序で文字 c の前に来る文字を表すものとする。ただし、辞書的順序で最小のアルファベット a に対しては、 $c[a - 1]$ は0を表すものとする。

【0038】

図3は、上記の検索アルゴリズムに対応するフローチャートである。同図を参照して、本実施の形態によるパターンの検索手順を説明する。この検索方法は、パターンを当該パターンの構成文字列の後ろから検索することが特徴である。図3に示す検索アルゴリズムにより、求めるパターンは、テキスト T の接尾辞配列 SA において、 $SA[j]$ ($start \leq j \leq end$) の位置から始まる場所に存在するので、それを列挙すればよい。

【0039】

図3を参照すると、検索部30は、まず、 $start$ に $C[P[m] - 1] + 1$ を代入し、 end に $C[P[m]]$ を代入し、 $start$ と end の値を求める。また、 $i = m - 1$ とする（ステップ301）。

次に、 i の値が正 ($i > 0$) かどうかを調べ、正であれば、次に、 $c = P[i]$ として、 $start$ に $C[c - 1] + f(start, c)$ を代入し、 end に $C[c - 1] + f(end, c)$ を代入し、 $start$ と end の値を求める。また、 $i = i - 1$ とする（ステップ302、303）。

次に、 end の値が $start$ の値を下回ったかどうかを調べ、下回ったならば、検索パターン P にマッチする文字列はテキスト T には存在しないことがわかるので、処理を終了する（ステップ304、305）。

一方、 end の値が $start$ の値を下回っていなければ、ステップ302に

戻って、新たな i について $start$ 及び end の値を求める（ステップ304）。

ステップ302において、 i の値が0以下になったならば、 $start$ 及び end の値を用い、 $start \leq j \leq end$ であるような全ての j に対して、 $SA[j]$ の位置から始まるテキスト T の接尾辞を出力して処理を終了する（ステップ306）。このとき、当該接尾辞と検索パターン P とがマッチする。

【0040】

次に、

$T[1 \dots 12] = "mississippi\$"$
 $P[1 \dots 3] = "ssi"$
 $SA[1 \dots 12]$
 $= \{8 \ 5 \ 2 \ 11 \ 1 \ 10 \ 9 \ 7 \ 4 \ 6 \ 3 \ 12\}$

の場合について、前処理部20によるデータ構造の構築及び検索部30によるパターン P の検索の動作例を説明する。

本動作例では、テキスト T を区切る基準として $k = 4$ とする。

【0041】

まず、 $k = 4$ の場合のテーブル F を作成する。

上述したように、 $F[i][c]$ には $f(k * i, c)$ が入る。そして、 $k * i < n + k$ であり、 $n = 12$ であるから、 $k = 4$ の場合、 i の値は1、2、3である。したがって、テーブル F には、 $i = 1, 2, 3$ 及び $c = 'i', 'm', 'p', 's', '$'$ の各々について、 $f(4 * i, c)$ の値が入り、図4に示すようになる。

例えば、 $F[2][p]$ の場合、 $f(4 * 2, 'p')$ であるから、配列 $B[1 \dots 12] = "ssmp\$pissi\$"$ において8 ($= 4 * 2$) 番目の文字である 's' 以前に 'p' は2個存在する。したがって、テーブル F の $F[2][p]$ には2が入る。なお、図4のテーブル F では、' \$' に対するエントリーも入れているが、実際には、検索パターンの中に '\$' が入ること

は考えなくて良いので、‘\$’に対する列は省略することができる。

【0042】

次に、テーブルLを作成する。

上述したように、 $h(i, c)$ を $f(4 * i, c) - f(4 * (i-1), c)$ とし、 $l(i, c)$ を、 $h(i, d)$ ($d < c$, 辞書順)の総和とすると、文字cの順序は‘i’<‘m’<‘p’<‘s’<‘\$’であるから、テーブルLは、図5に示すようになる。

例えば、 $L[2][‘s’]$ は、 $h(2, ‘i’)$ と $h(2, ‘m’)$ と $h(2, ‘p’)$ との総和であり、図4のテーブルFを参照すれば、

$$\begin{aligned} h(2, ‘i’) &= f(4 * 2, ‘i’) - f(4 * (2-1), ‘i’) \\ &= f(8, ‘i’) - f(4, ‘i’) = 1 - 0 = 1 \end{aligned}$$

すなわち、配列B [5 · · · 8]において、‘i’が一つ存在することがわかる。同様に、

$$h(2, ‘m’) = f(8, ‘m’) - f(4, ‘m’) = 1 - 1 = 0$$

$$h(2, ‘p’) = f(8, ‘p’) - f(4, ‘p’) = 2 - 1 = 1$$

したがって、テーブルLの $L[2][‘s’]$ には2 (= 1 + 0 + 1) が入る。

【0043】

ところで、図4及び図5を参照すると、テーブルLにおいて、

$$L[i][c+1] = L[i][c] + F[i][c] - F[i-1][c]$$

という関係がある。ただし、 $c+1$ は辞書的順序で文字cの次にくる文字である。また、 $F[0][c] = 0$ としている。例えば、上述した $L[2][‘s’]$ の場合、

$$L[2][s]$$

$$= L[2][p] + F[2][p] - F[1][p]$$

$$= 1 + 2 - 1 = 2$$

となる。このことから、 x 個おきの文字に対してのみテーブルLを作成し、間の文字に対する値はテーブルL及びテーブルFから算出することにより、メモリを節約することができる。ただし、この場合、この部分の計算時間は x 倍となる。なお、図4の場合と同様に、テーブルLにおいても‘\$’の列は省略することができる。

【0044】

次に、テーブルGを作成する。

上述したように、全ての r ($0 < 4 * r < n + 4$) に対して、 $T[SA[q + 4 * (r - 1)] - 1]$ (ただし、 $0 < q \leq 4$) の値が同じ物ごとに辞書的順序にしたがって並べ替えたものがテーブル $G[r][1 \dots 4]$ である (ただし、 r の値が上述した数1を満足する値である場合は $0 < q \leq n - (r - 1) * 4$)。ここで、 $[SA[q + 4 * (r - 1)] - 1]$ は、配列Bにおいて、 $B[1 \dots 4]$ 、 $B[5 \dots 8]$ 、 $B[9 \dots 12]$ に対応する。したがって、例えば $G[1][1 \dots 4]$ は、 $B[1 \dots 4] = "s s m p"$ であるから対応する $q = 1, 2, 3, 4$ を‘s’‘s’‘m’‘p’の辞書的順序で並べ替えれば、

$$G[1][1 \dots 4] = \{3, 4, 1, 2\}$$

となる ($q = 1$ の‘s’と $q = 2$ の‘s’については、 q の小さい方を先にしている)。同様に、 $r = 2, 3$ についても考え、結果として、

$$G[1 \dots 3][1 \dots 4]$$

$$= \{3, 4, 1, 2\}, \{3, 2, 4, 1\}, \{2, 3, 4, 1\}$$

を得る。

【0045】

次に、以上のテーブルF、L、Gを用いて計算される $g(i, c, j)$ 、 $f'(j, c)$ 及び $f(j, c)$ について、具体的な算出例を挙げる。まず、 $g(3, 'r', 2)$ について、

$$\begin{aligned} g(3, 'r', 2) &= G[3][L[3]['i'] + 2] + 4 * (3 - 1) \\ &= G[3][0 + 2] + 8 = 11 \end{aligned}$$

となる。

また、 $f'(10, 'i')$ を求めるには、 $g(3, 'i', x)$ ($0 < x \leq 3$) の中から 10 以下の値を取る最大の x を求めれば良い。上記と同様に $g(3, 'r', 1)$ 、 $g(3, 'r', 3)$ を求めると、

$$\begin{aligned} g(3, 'r', 1) &= 10 \\ g(3, 'r', 3) &= 12 \end{aligned}$$

であるから、 $f'(10, 'i') = x = 1$ が得られる。

さらに、 $f(10, 'i')$ の値は、

$$f(10, 'i') = F[2]['i'] + f'(10, 'i') = 1 + 1 = 0$$

と求まる。

【0046】

次に、上記のデータ構造を用いて、

$$P[1 \dots 3] = "ssi"$$

の検索を行う。

図3のフローチャートに示したアルゴリズムにおいて、まず、startに

$$C[P[3]-1] + 1 = C[i-1] + 1 = 0 + 1 = 1$$

が代入され、endに

$$C[P[3]] = C[i] = 4$$

が代入される（ステップ301参照）。これは、検索のための中間パターンである $P[3] = "i"$ が、テキストTに対する接尾辞配列SAのどの範囲に位置しているかを示す。すなわち、 $SA[1 \dots 4] = \{8 \ 5 \ 2 \ 11\}$ に対応するテキストTの要素（テキストTの8番目と5番目と2番目と11番目の要素）が中間パターン“i”と一致する。

【0047】

次に、 $i = 2 (= 3 - 1) > 0$ であるので（ステップ301参照）、cに $P[i]$ が代入される（ステップ302、303参照）。そして、start及びendに代入される値を計算する。すなわち、

start

$$\begin{aligned} &= C[P[2]-1] + f(start, P[2]) \\ &= C['s'-1] + f(1, 's') = C['p'] + f(1, 's') \end{aligned}$$

ここで、 $f(1, 's')$ は、 $k * i = 1$ なのでテーブルFから直接は求められず、

$$\begin{aligned} &C['p'] + f(1, 's') \\ &= 7 + F[1-1] ['s'] + f'(1, 's') \end{aligned}$$

ここで、 $F[0][c] = 0$ であり、 $f'(1, 's')$ は、 $g(1, 's', x)$ で $x = 1$ の時に

$$\begin{aligned} & g(1, 's', 1) \\ & = G[1][L[1]['s'] + 1] + 4 * (1 - 1) = G[1][2 + 1] \\ & = 1 \end{aligned}$$

となるので、 $f'(1, 's') = 1$ である。したがって、

$$C['p'] + f(1, 's') = 7 + 0 + 1 = 8$$

となり、`start`には8が代入される。また、

$$\begin{aligned} & \text{end} \\ & = C[P[2] - 1] + f(\text{end}, P[2]) \\ & = C['p'] + f(4, 's') \end{aligned}$$

ここで、 $f(4, 's')$ は、 $k * i = 4 * 1$ なので、テーブルFから直接求められ、 $F[1]['s'] = 2$ であるから、

$$C['p'] + f(4, 's') = 7 + 2 = 9$$

となり、`end`には9が代入される。これは、検索のための中間パターンである $P[2..3] = "s i"$ が、テキストTに対する接尾辞配列SAのどの範囲に位置しているかを示す。すなわち、 $SA[8] = \{7\}$ と $SA[9] = \{4\}$ とに対応するテキストTの要素（テキストTの7番目と4番目の要素）から始まる要素数2のパターンが中間パターン“`s i`”と一致する。

【0048】

次に、`end` (=9) > `start` (=8) であるからステップ302に戻り

(ステップ304参照)、 $i = 1 (= 2 - 1) > 0$ であるので再度ステップ303に進み、 c に $P[i]$ が代入される(ステップ302参照)。そして、 $start$ 及び end に代入される値を計算する。すなわち、

$start$

$$\begin{aligned} &= C[P[1] - 1] + f(start, P[1]) \\ &= C['s' - 1] + f(8, 's') = C['p'] + f(8, 's') \end{aligned}$$

ここで、 $f(8, 's')$ は、 $k * i = 4 * 2$ なので、テーブルFから直接求められ、 $F[2]['s'] = 3$ であるから、

$$C['p'] + f(8, 's') = 7 + 3 = 10$$

となり、 $start$ には10が代入される。また、

end

$$\begin{aligned} &= C[P[1] - 1] + f(end, P[1]) \\ &= C['p'] + f(9, 's') \end{aligned}$$

ここで、 $f(9, 's')$ は、 $k * i = 11$ なのでテーブルFから直接は求められず、 $4 * (i - 1) < 9 \leq 4 * i$ から $i = 3$ であるから、

$$\begin{aligned} &C['p'] + f(9, 's') \\ &= 7 + F[3 - 1]['s'] + f'(9, 's') \end{aligned}$$

ここで、 $F[2]['s']$ はテーブルFから3、 $f'(9, 's')$ は、 $g(3, 's', x)$ で $x = 1$ の時に

$$g(3, 's', 1)$$

$$\begin{aligned}
 &= G[3][L[3]['s'] + 1] + 4 * (3 - 1) \\
 &= G[3][3 + 1] + 8 = 9
 \end{aligned}$$

で、解はこれだけなので、 $f'(9, 's') = x = 1$ である。したがって、

$$C['p'] + f(9, 's') = 7 + 3 + 1 = 11$$

となり、`end` には 11 が代入される。

【0049】

次に、`end` ($= 11$) $>$ `start` ($= 10$) であるからステップ302に戻り（ステップ304参照）、 $i = 0$ ($= 1 - 1$) となったので（ステップ302参照）、`start` $\leq j \leq end$ であるような全ての j に対して、`SA[j]` の位置から始まるテキスト T の接尾辞を求める（ステップ306参照）。ここでは、`start = 10`、`end = 11` であるから、`SA[10] = 6`、`SA[11] = 3` であり、`T[3 · · · 5] = T[6 · · · 8] = "ssi"` となっており、パターン P と一致している。

【0050】

上記の動作例では、 $k = 4$ である場合について説明したが、 k の値は、検索対象であるテキスト T の文字数 (n)、検索パターン P の文字数 (m)、テキスト T を構成するアルファベットにおける文字の種類の数 (s) などに応じて適宜に設定することができる。この場合、 k の値に応じて、上述した前処理及び検索処理に必要なメインメモリ 103 の記憶容量とこれらの処理に要する時間とが変化する。大まかには $k = O(s)$ 、すなわち s の定数倍とすると、メインメモリ 103 に必要な記憶容量は $O(n \log n)$ ビット、検索時間は $O(m \log s)$ となり、従来の接尾辞木を用いる検索方法における理論値と同じである。例えば $k = s$ のとき、 $3n \log s + 2n \log n$ ビットが必要となる。ただし、この場合が最小であるわけではない。

実際には、メインメモリ 103 の記憶容量は、8 ビット、16 ビット、32 ビットの倍数（場合によっては約数）であることがほとんどなので、このことを考

慮して k を設定することが好ましい。

【0051】

次に、具体的なテキスト T に対して本実施の形態を適用した場合におけるメインメモリ 103 に必要な記憶容量（データサイズ）と検索時間とを例示する。

【適用例 1】

文字が 1 バイトで表され、256 種類である場合（終了判定文字 \$ も同時に表したい場合は 255 種類）。通常の英文テキストなどがこれに該当する。

この場合、テキスト T の文字数を n とすれば、テキスト T のサイズは n バイト、接尾辞配列 SA のサイズは $4n$ バイトである。

例えば、 $k = 65536$ ($= 2^{16}$) とすると、 k 以下の数字は 2 バイトで表すことができる。これにより、上述したテーブル F 、 L 、 G 、 C の合計サイズは、 $2n$ バイト強となる。したがって、テキスト T 及び接尾辞配列 SA 、テキスト T を含んだデータサイズでも $7n$ バイト強である。これは当該テキスト T に対する接尾辞木のサイズ ($20n \sim 40n$ バイト程度) の 3 分の 1 程度である。

一方、検索速度は、 $\log k$ に比例するので、 k を小さくすると速度を上げることが可能である。

例えば、 $k = 256$ ($= 2^8$) とすると、 $k = 65536$ の場合に対して 2 倍の検索速度を見込める。この場合、テーブル F 、 L 、 G 、 C を持つために必要なメインメモリ 103 の記憶容量は $6n$ バイトである。すなわち、テキスト T 及び接尾辞配列 SA を加えた総量でも $11n$ バイトのデータサイズとなり、やはり接尾辞木よりも小さい。

【0052】

【適用例 2】

文字が 2 バイトで表され、 65536 ($= 2^{16}$) 種類ある場合。日本語のテキストなどがこれに該当する。

この場合、 $k = 65536$ とすると、テーブル F 、 L 、 G 、 C の合計サイズは、 $8n$ バイトであり、テキスト T 及び接尾辞配列 SA を加えた総量でも $14n$ バイトである。

なお、この例の場合、 $k = 256$ などの小さい値とすることは、データサイズが

大きくなってしまうので現実的ではない。

【0053】

【適用例3】

DNAの配列（文字の種類数は4）の場合。

2bitの文字、4bitの文字を扱うことを許すならば、 $k = 4$ の場合、テーブルF、L、G、CとテキストT及び接尾辞配列SAとを加えた総データサイズは8.75nバイト程度となる。また、 $k = 16$ の場合、総データサイズは5.375nバイト程度となる。特に後者の場合、接尾辞配列SAそのものとほとんど変わらないデータサイズとなっている。

【0054】

次に、実際のDNA配列に対する検索速度の測定例を示す。

この測定例では、本実施の形態による検索方法と、接尾辞配列SAを2分探索する従来の検索方法とを用いて、大腸菌の全配列に対し、同じクエリーを10000000回繰り返した場合の計算時間を比較している。なお、計算機は、CPUが333MHz Power PCのRS6000（米国IBM社のワークステーション）である。

検索パターンP = “CACATAA”

本実施の形態による検索時間 : 0.38秒

従来の2分探索による検索時間 : 4.30秒

検索パターンP = “AGAGCGGC”

本実施の形態による検索時間 : 0.47秒

従来の2分探索による検索時間 : 4.02秒

検索パターンP = “CCCGCTTCGGC”

本実施の形態による検索時間 : 0.76秒

従来の2分探索による検索時間 : 3.35秒

検索パターンP = "ACCGCGAAATACCGGCGCGAAATCATCGACTTACGCATAGGCAC"

本実施の形態による検索時間 : 3. 13秒

従来の2分探索による検索時間 : 3. 88秒

検索パターンP = "CGGCGTCAGGTACTGACCGCGACCAATGCGA"

本実施の形態による検索時間 : 0. 84秒

従来の2分探索による検索時間 : 3. 41秒

以上のように、全ての例において、本実施の形態の方が2分探索よりも計算時間が短縮（高速化）されている。最も高速化されている例（検索パターンP = "AGAGCGGC"）では10倍以上高速になっている。また、短い配列のクエリーほど高速化の効果があることがわかる。

【0055】

なお、本実施の形態では、テキストTの接尾辞配列SAを探索して所望のパターンPを検索する場合について説明したが、テキストTの接頭辞配列を探索してパターンPを検索することも可能である。

ここで、接頭辞とは、所定の文字列において、所定の文字を特定した場合の当該文字以前の文字列である。この接頭辞に対して、接尾辞に対する接尾辞木と同様の接頭辞木を生成することができる。また、接頭辞配列とは、テキストTにおける全ての接頭辞を後から順に並べた文字列を、辞書的順序で並べ替えた場合のインデックスの配列である。すなわち、文字列の先頭方向と末尾方向（左右）を逆にしたテキストTに対する接尾辞配列と本質的に同じである。したがって、方向を考慮することにより、上述した手法をそのまま接頭辞配列に対しても用いることができる。

【0056】

【発明の効果】

以上説明したように、本発明によれば、大規模テキストデータベースの検索に

おいて、処理を行うためのデータ構造におけるデータサイズの増大を抑えながら、高速な検索を実現することができる。

【図面の簡単な説明】

【図1】 本実施の形態を実現するのに好適なコンピュータ装置のハードウェア構成の例を模式的に示した図である。

【図2】 本実施の形態におけるデータ構造の構築及び検索を行うための機能ブロックを示す図である。

【図3】 本実施の形態におけるパターンの検索アルゴリズムを説明するフローチャートである。

【図4】 本実施の形態において用いられるテーブルFの構成例を示す図である。

【図5】 本実施の形態において用いられるテーブルLの構成例を示す図である。

【図6】 接尾辞木の例を示す図である。

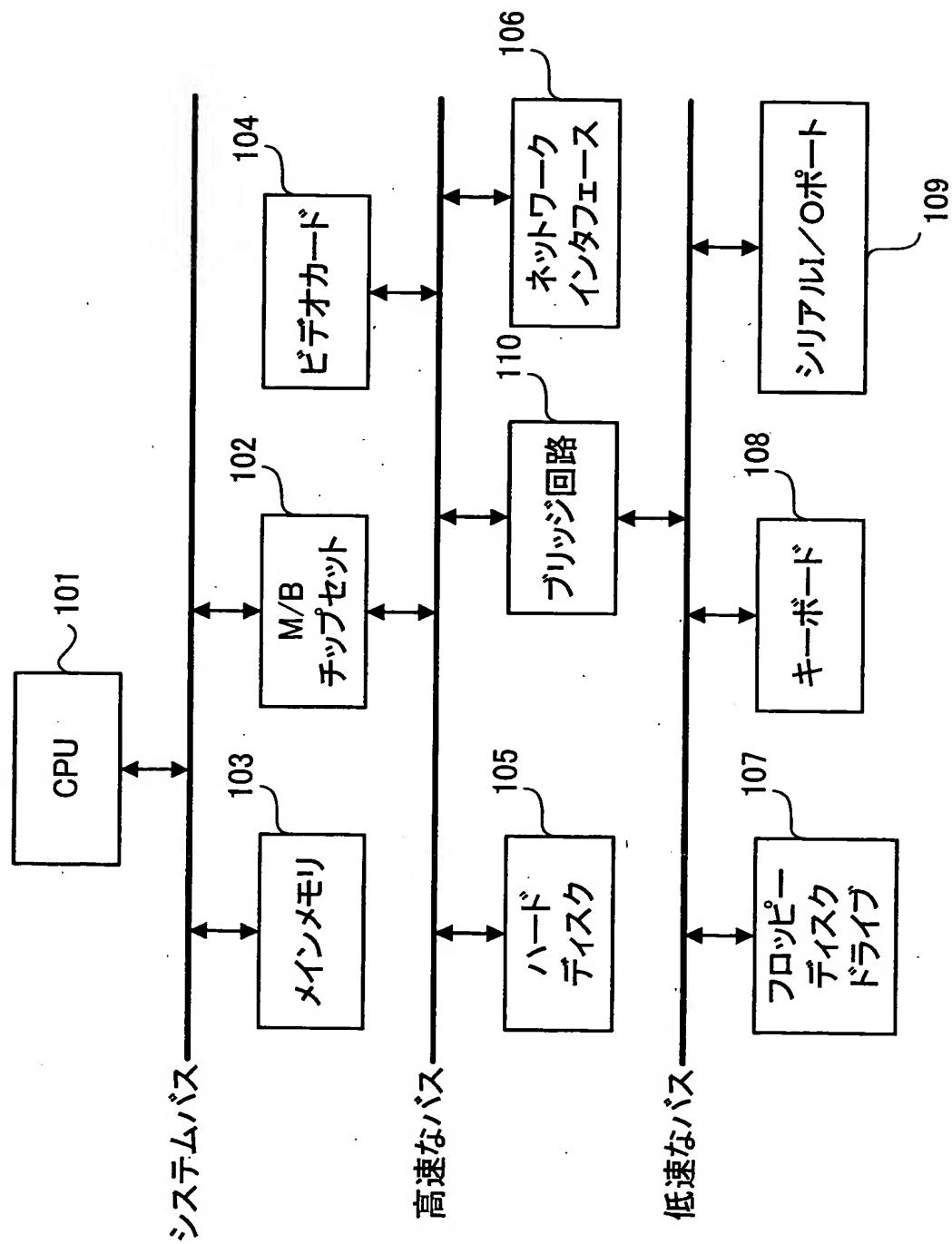
【符号の説明】

10…接尾辞配列生成部、20…前処理部、30…検索部、101…CPU（中央処理装置）、102…M/B（マザーボード）チップセット、103…メインメモリ、104…ビデオカード、105…ハードディスク、106…ネットワークインターフェース、107…フロッピーディスクドライブ、108…キーボード、109…シリアルI/Oポート、110…ブリッジ回路

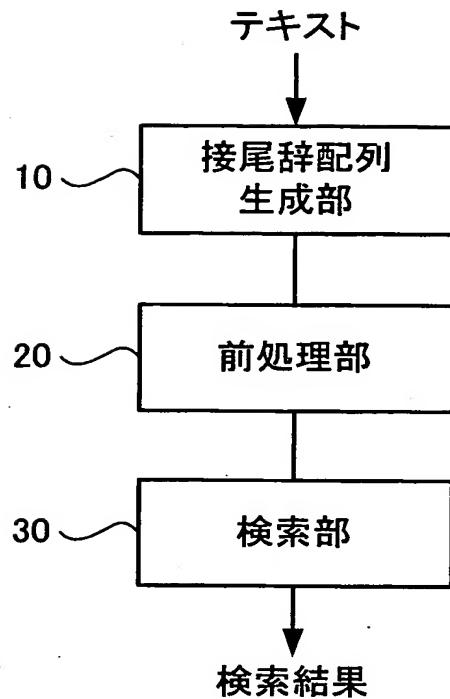
【書類名】

図面

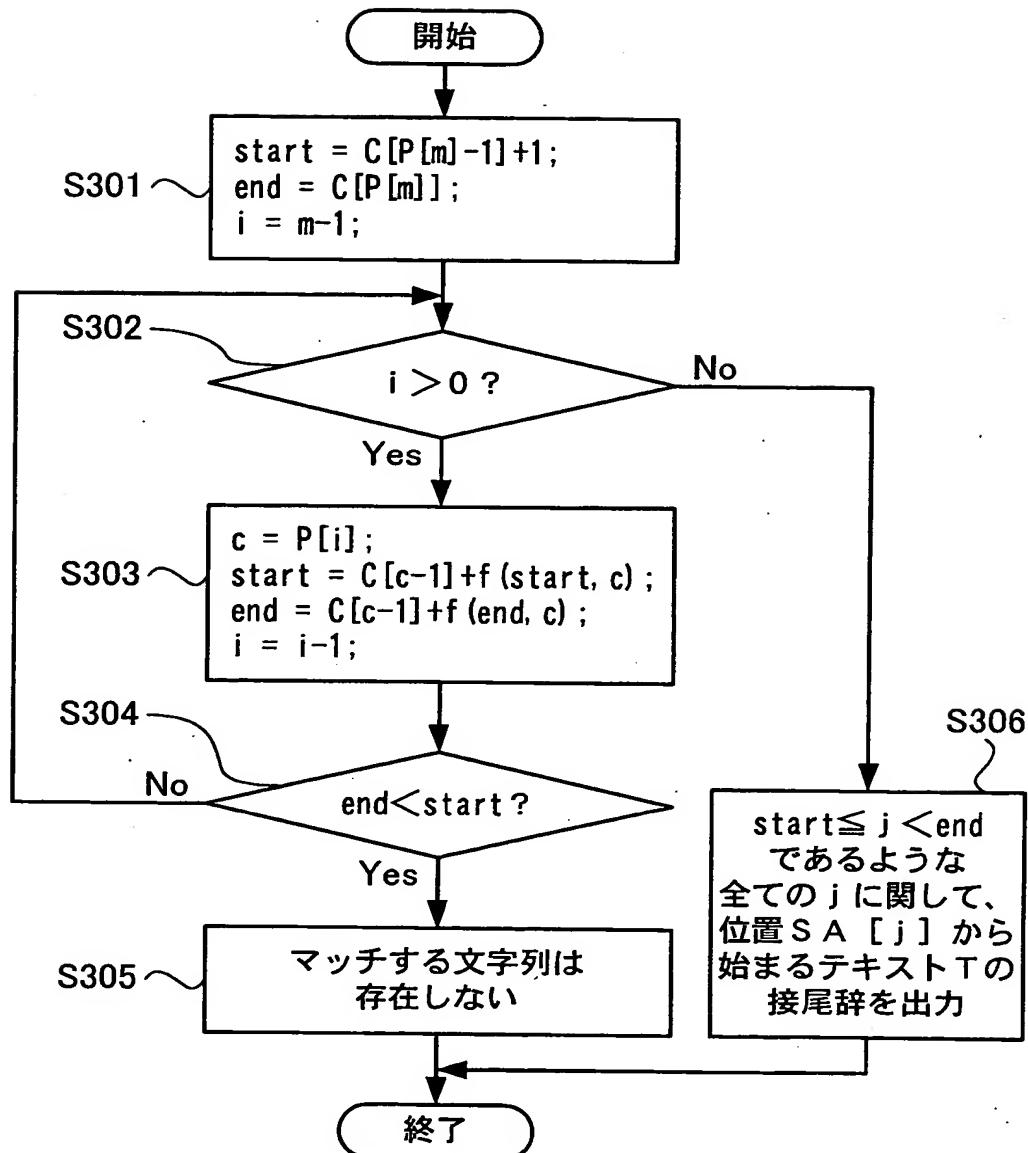
【図1】



【図2】



【図3】



特2001-004189

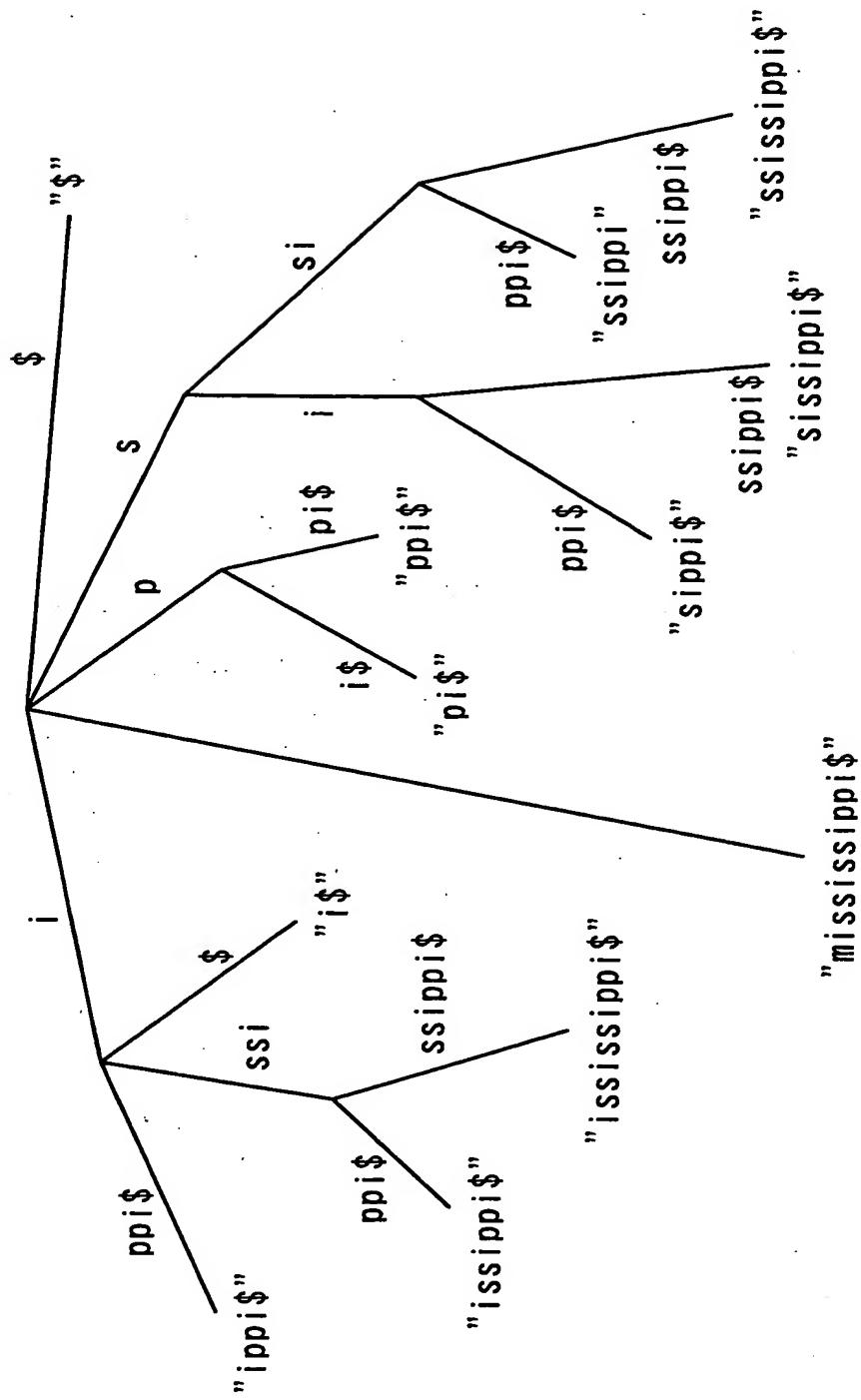
【図4】

i, c	i	m	p	s	\$
1	0	1	1	2	0
2	1	1	2	3	1
3	4	1	2	4	1

【図5】

i, c	i	m	p	s	\$
1	0	0	1	2	4
2	0	1	1	2	3
3	0	3	3	3	4

【図6】



【書類名】 要約書

【要約】

【課題】 大規模テキストデータベースの検索において、処理を行うためのデータ構造におけるデータサイズの増大を抑えながら、高速な検索を実現する。

【解決手段】 検索対象である文字列中から所望のパターンを検索するパターン検索方法において、次の範囲検索ステップと、文字列抽出ステップとを含む。すなわち、範囲検索ステップにおいて、このパターンの最後の文字から前方へ1文字ずつ順に加えて得られる各中間パターンに関して、この中間パターンの先頭の文字が検索対象の文字列に対する接尾辞配列のどの範囲に存在するかを順次検索する。次に、文字列抽出ステップにおいて、当該接尾辞配列の範囲に含まれる各要素に対応する文字列の要素を特定し、この文字列の各要素を先頭としてこのパターンの要素数と同じ数の要素からなる部分文字列を抽出する文字列抽出ステップとを含むことを特徴とする。

【選択図】 なし

認定・付加情報

特許出願の番号	特願2001-004189
受付番号	50100029411
書類名	特許願
担当官	末武 実 1912
作成日	平成13年 2月27日

<認定情報・付加情報>

【特許出願人】

【識別番号】	390009531
【住所又は居所】	アメリカ合衆国10504、ニューヨーク州 アーモンク (番地なし)
【氏名又は名称】	インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】

【識別番号】	100086243
【住所又は居所】	神奈川県大和市下鶴間1623番地14 日本アイ・ビー・エム株式会社 大和事業所内
【氏名又は名称】	坂口 博

【代理人】

【識別番号】	100091568
【住所又は居所】	神奈川県大和市下鶴間1623番地14 日本アイ・ビー・エム株式会社 大和事業所内
【氏名又は名称】	市位 嘉宏

【代理人】

【識別番号】	100106699
【住所又は居所】	神奈川県大和市下鶴間1623番14 日本アイ・ビー・エム株式会社大和事業所内
【氏名又は名称】	渡部 弘道

【復代理人】

【識別番号】	100104880
【住所又は居所】	東京都港区赤坂5-4-11 山口建設第2ビル 6F セリオ国際特許事務所
【氏名又は名称】	吉部 次郎

【選任した復代理人】

【識別番号】	100100077
--------	-----------

次頁有

特2001-004189

認定・付加情報（続き）

【住所又は居所】 東京都港区赤坂5-4-11 山口建設第2ビル
6F セリオ国際特許事務所
【氏名又は名称】 大場 充

次頁無

出願人履歴情報

識別番号 [390009531]

1. 変更年月日 2000年 5月16日

[変更理由] 名称変更

住 所 アメリカ合衆国10504、ニューヨーク州 アーモンク (番地なし)

氏 名 インターナショナル・ビジネス・マシーンズ・コーポレーション